AMENDMENTS TO THE SPECIFICATION:

Please replace paragraph **[0023]** with the following amended paragraph:

The system 101 may communicate with other computers or networks of computers by way of a network adapter, represented at 108, capable of communicating with a network 109. Example network adapters are communications channels, token ring, Ethernet or modems. Alternatively, the workstation 101 may communicate using a wireless interface, such as a CDPD (cellular digital packet data) card. The workstation 101 may be associated with such other computers in a local area network (LAN) or a wide area network (WAN), or the workstation 101 can be a client in a client/server arrangement with another computer, etc. All of these configurations, as well as the appropriate communications hardware and software, are known in the art.

Please replace paragraph **[0024]** with the following amended paragraph:

FIG. 2 illustrates a data processing network 200 in which the present invention may be practiced. The data processing network 200 may include a plurality of individual networks, such as wireless network and a wire network, each of which may include a plurality of individual workstations ~~101~~ 201, 202, 204. Additionally, as those skilled in the art will appreciate, one or more LANs may be included, where a LAN may comprise a plurality of intelligent workstations coupled to a host processor.

Please replace paragraph **[0025]** with the following amended paragraph:

Still referring to FIG. 2, the networks may also include mainframe computers or servers, such as a gateway computer (client server 206) or application server (remote server 208 which may access a data repository) connected to clients 201, 202, 203, 204. A gateway computer 206 serves as a point of entry into each network 207. A gateway is needed when connecting one networking protocol to another. The gateway 206 may be preferably coupled to another network (the Internet 207 for example) by means of a communications link. The gateway 206 may also be directly coupled to one or more workstations ~~101~~ 201, 202, 203 using a communications link.

The gateway computer may be implemented utilizing an IBM eServer, zServer, 900 Server available from IBM. Figure 2 also shows a remote server 208 connected to Internet 207 and to client 205 for user 211.

Please replace paragraph **[0028]** with the following amended paragraph:

In the preferred embodiment, the present invention is implemented as one or more computer software programs 111. The implementation of the software of the present invention may operate on a user's workstation, as one or more modules or applications 111 (also referred to as code subroutines, or "objects" in object-oriented programming) which are invoked upon request. Alternatively, the software may operate on a server in a network, or in any device capable of executing the program code implementing the present invention. The logic implementing this invention may be integrated within the code of an application program 112, or it may be implemented as one or more separate utility modules which are invoked by that application, without deviating from the inventive concepts disclosed herein. The application ~~111~~ 112 may be executing in a Web environment, where a Web server provides services in response to requests from a client connected through the Internet. In another embodiment, the application may be executing in a corporate intranet or extranet, or in any other network environment. Configurations for the environment include a client/server network, Peer-to-Peer networks (wherein clients interact directly by performing both client and server function) as well as a multi-tier environment. These environments and configurations are well known in the art.

Please replace paragraph **[0031]** with the following amended paragraph:

FIG. 3C depicts yet another embodiment where the application code and the XRT module 324 execute on an application server 322 separate from the database server 323 that hosts the database 325. The user accesses the results, via network 326, in the same fashion as the embodiment in FIG. 3B from a user workstation 321 using the client code 324 that might be an application specific code or a general purpose browser that interfaces to the application. The application code and XRT module 324 accesses data from the database 325 through JDBC communication on the network 327. In this case as in the case of FIG. 3A, the application code and XRT 324 might access more than one database server 323.

Please replace paragraph **[0035]** with the following amended paragraph:

An example Relational Database schema in FIG. 4 shows two tables 401 and 402 in the database. Table 401 stores information about a purchase order (PO) and uses a primary key (POID) 410 as the primary identifier for each record. Table 402 stores information about a particular line item (LINEITEM) which was purchased. The relationship between records in each of these tables is described using the primary key POID 410. A complete purchase order record would require a query to table 401 to retrieve the buyer information (411, 413, 414) and seller information (412, 415), using a join 430 to table 402 to retrieve information (420, 421, 422, 423, 424, 425, 426, 427) about each item purchased. This join relationship inherently produces a tree structure of queries, where an initial query on table 401 will result in a key 410 which is in turn used in subsequent queries to retrieve the appropriate results from table 402. XRTL provides the necessary constructs to specify the mapping of the records/columns of the relational table model onto the elements/attributes of the XML object and imposing a hierarchical structure on the output. XRT supports insert, update and delete Structured Query Language (SQL) functions for mapping an XML object onto a relational database, for operations within a tree-structured transactional framework (SQL is a language used to retrieve data from databases). It also specifies the source or sink databases that are used during the transformation. XRT supports the specification of multiple heterogeneous data sources and sinks, where each source or sink can be a file, a database, or an Internet data stream for example. XRTL is used to create an XRT script. An XRT script specifies the sources/sinks of the transformations, the mapping between the XML constructs and the relational database constructs, as well as additional information such as transactional groupings or structural patterns. Examples of such a script can be found in TABLE 2.

Please replace paragraph **[0038]** with the following amended paragraph:

The template that defines the shredding, or data insertion, process is also represented as a tree structure, called the Transaction Tree. An example Transaction Tree specification is shown in FIG. 6 corresponding to the shredding script in TABLE 3. The structure of the Transaction Tree specifies the sequence of operations that insert the XML objects into the database. Each Transaction may have one or more sub-Transactions and one or more Store actions. For example Transaction Node 601 corresponding to lines C6-C16 in Table 3 has a sub-transaction 602 corresponding to lines C12-C15 and an insert action 603 corresponding to lines C7-C11. Figure 6 also shows a transaction 604.

Please replace paragraph **[0050]** with the following amended paragraph:

The XRT run-time transformation engine (701) consists of 4 main components:
The XRT parser (701a) that reads input XRT scripts (702) and creates the corresponding Run-Time Object (704);

The Cache (701b) where the Run-Time Objects are stored for multiple use. The usage pattern that has been identified is that the same XRT script is used multiple times with different External Parameters (703). Caching the Run Time Object (704) provides a significant speed-up, since it avoids parsing the same XRT script (702) more than once;

The actual transformation procedure starts in the XRT Reader (701c), that executes the retrieve instructions as specified in the Run-Time Object and uses as input an optional External Parameter File (703) and the database or databases (705) specified in the Run-Time Object; It creates a sequence of SAX events (706) that are passed to the XSL Transformer; and

The XSL transformer (701d) absorbs the SAX events generated by the XRT Reader and transforms the logical XML specified by this sequence of events based on the XSL instructions in the Run-Time object (704) to create the Output XML (707)

Please replace paragraph **[0052]** with the following amended paragraph:

The detailed steps of the retrieval procedure are as follows (refer to FIG. 8A and 8B, in conjunction with FIG. 7):

In Step (801), the XRT run-time transformation engine (701) is invoked to process a certain retrieval operation with a a particular XRT Script (702) specified with an URA and an optional External Parameter File (703);

In Step (802), the engine checks whether the specified XRT Script has been already processed and the corresponding Run-Time Object exists in the Cache;

If the Run-Time Object does not exists in cache, the XRT Script is loaded through the specified URA and parsed in step 804 by the parser (701a) and the corresponding Search Tree and XSL Template created (Steps 805 and 806);

The created objects are stored in the cache with the XRT Script URA as the entry key (Step 807);

Whether the run-time object was in the cache originally, or was created and cached during the current invocation, it is retrieved from the cache in step 803;

In Step 851, the system checks whether an external parameters file was specified for this invocation, and if so parses it (Step 852);

In Step 853, the XRT reader (701c) of the engine runs the Search Tree (704a) and executes the SQL commands from the Tree (Step 853a), using the external parameters (703) as parameters to the SQL commands, if appropriate;

As a result, information is retrieved from the database (Step 853b) via JDBC and the results are passed to the XSL transformer as SAX events (Step 853c);

In Step 653d, the SAX events are fed to the XSL Transformer (501d) and the XSL template (704b) of the stored object is applied, to build the final XML; and

Steps 853a, 853b, 853c and 853d are executed as long as the search tree has more queries, and step 854 is to output the final XML.

Please replace paragraph **[0057]** with the following amended paragraph:

The detailed steps of the shredding procedure are as follows (refer to FIG. 10A, 10B and 10C, in conjunction with FIG. 9):

In Step (1001), the XRT run-time transformation engine (901) is invoked to process a certain shredding operation with a particular XRT Script (902) specified with an URA and an Input XML document (903);

In Step (1002), the engine checks whether the specified XRT Script has been already processed and the corresponding Run-Time Object exists in the Cache;

If the Run-Time Object does not exists in cache, the XRT Script is loaded through the specified URA and parsed in step 1004 by the parser (901a) and the corresponding Transaction Tree and XSL Template are created (Steps 1005 and 1006);

The created objects are stored in the cache with the XRT Script URA as the entry key (Step 1007);

Whether the run-time object was in the cache originally, or was created and cached during the current invocation, it is retrieved from the cache in (step 1003);

In Step ~~851~~ 1051, the system checks whether external parameters were specified for this invocation, and if so it passes them to the XSL Transformer (Step 1052);

In Step 1053, the XSL Transformer (901c) of the engine starts reading in the Input XML (902) and applying the XSL templates specified in the XSL Templates object (904a);

As each XSL template is applied to the Input XML (Step 1054), SAX Events are generated (Step 1055);

The SAX Events are passed to the shredding Tree that transforms the SAX Events into database records (Step 1056);

As long as there are XSL templates to apply, at step 1057, Steps 1054, 1055 and 1056 are repeated;

When, at step 1057, all XSL templates are applied, the Transaction Tree starts executing the database operations in the order and sequence specified within the tree (Step 1081);

If, at step 1050, a particular database operation fails, all database operations specified within the scope of a transaction are rolled back (Step 1083);

If all the database operations specified within the scope of a transaction are completed successfully, all these operations are committed (Step 1084); and